**ОТЧЁТ**

**ПО ЛАБОРАТОРНОЙ РАБОТЕ №9**

**Уравнение параболического типа**

**(Вариант 9)**

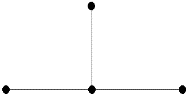
*Выполнил студент 3 курса МОиАИС*

***Постановка задачи:*** Решить параболическое уравнение

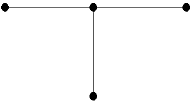


явным и неявным методами.

Шаблон для явного метода:



Шаблон для неявного метода:



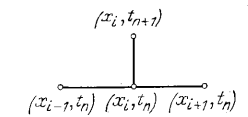
Вывести результаты в виде графиков U(x) для разных значений t от 0 до 10 c шагом 1.

Для всех вариантов [a, b] = [0; 1], [c, d] = [0; 10], D=1.

Погрешность решения 0,01.

***Результаты расчетов***

**Явный метод:**



* явная схема

Уравнение теплопроводности:

Введём схему, состоящую из четырёх узлов. Разностное уравнение, построенное по данной схеме:

Разрешая уравнение относительно :

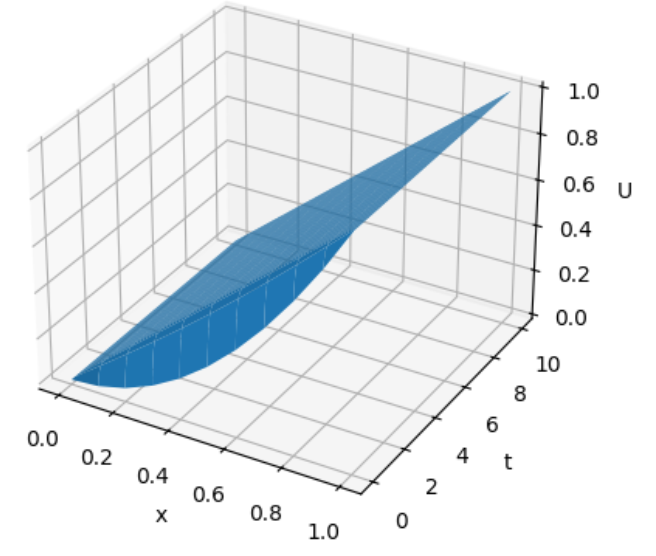
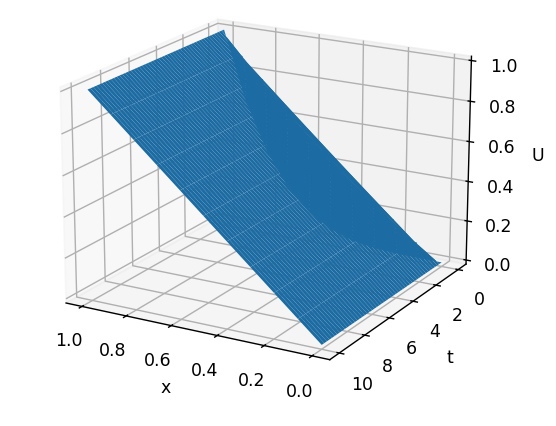
Начальные условия:

Граничные условия в сеточном виде:

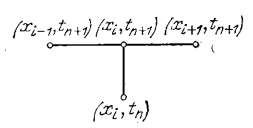
Рассмотренная разностная схема условно устойчива. Необходимое и достаточное условие устойчивости:

Возьмем в качестве первого шага разбиения: и .

Итерации завершаются при выполнении условия:



**Неявный метод:**



- неявная схема

Построим простейшую неявную схему. Разностное уравнение, построенное по данной схеме:

Из этого разностного соотношения можно получить систему уравнений относительно неизвестных значений сеточной функции на слое:

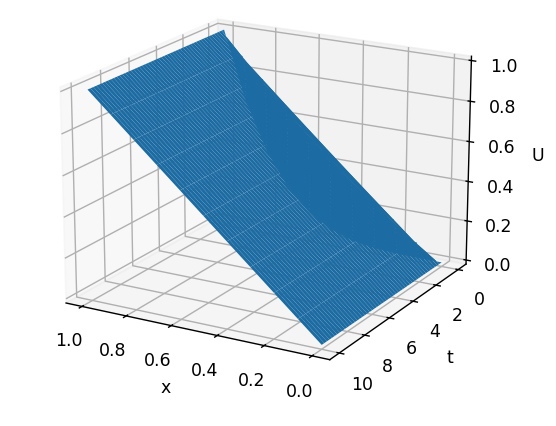
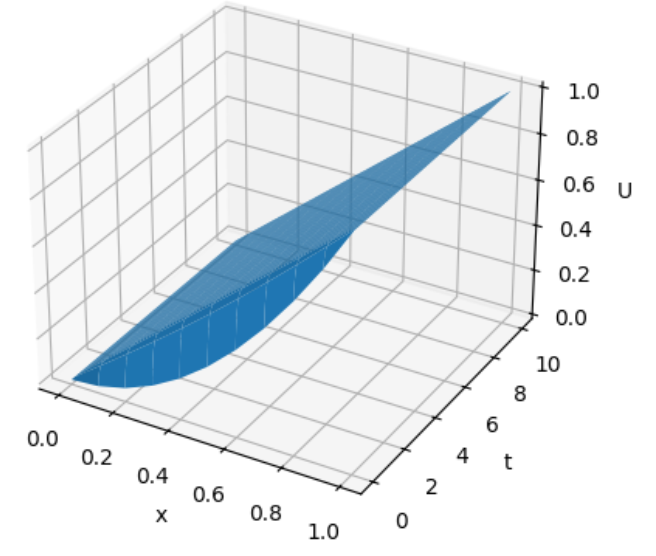
Полученная схема устойчива и сходится со скоростью .

Начальные условия:

Граничные условия в сеточном виде:

Возьмем в качестве первого шага разбиения: и . Итерации завершаются при выполнении условия:

Систему линейных уравнений решаем методом прогонки.



**ПРИЛОЖЕНИЕ**

*main.py*

import math  
import sympy as sy  
import numpy as np  
import matplotlib.pyplot as plt  
  
  
# Функция для визуализации результатов  
def printRes(res):  
 iSize = np.size(res, 0)  
 jSize = np.size(res, 1)  
 xValue = [0] \* iSize \* jSize  
 tValue = [0] \* iSize \* jSize  
 uValue = [0] \* iSize \* jSize  
  
 for i in range(iSize):  
 for j in range(jSize):  
 xValue[j + i \* jSize] = res[i, j][0]  
 tValue[j + i \* jSize] = res[i, j][1]  
 uValue[j + i \* jSize] = res[i, j][2]  
  
 x = np.reshape(xValue, (iSize, jSize))  
 t = np.reshape(tValue, (iSize, jSize))  
 u = np.reshape(uValue, (iSize, jSize))  
  
 fig = plt.figure()  
 ax = fig.add\_subplot(111, projection='3d')  
 ax.plot\_surface(x, t, u, alpha=0.7)  
  
 ax.set\_xlabel('x')  
 ax.set\_ylabel('t')  
 ax.set\_zlabel('U')  
  
 plt.show()  
 return  
  
  
# Явная схема  
def explicitSchema(u0x, ut0, ut1, aArg, hArg):  
 x = sy.Symbol('x')  
 t = sy.Symbol('t')  
 tauArg = hArg \*\* 2 / (2 \* aArg + 3)  
 tSize = math.floor(10 / tauArg) + 1  
 xSize = math.floor(1 / hArg) + 1  
 res = np.zeros((tSize, xSize), dtype='f,f,f')  
 lam = aArg \* tauArg / hArg \*\* 2  
  
 for i in range(xSize):  
 xValue = i \* hArg  
 uValue0 = u0x.subs(x, xValue)  
 res[0, i] = (xValue, 0.0, uValue0)  
  
 xValueLast = (xSize - 1) \* hArg  
  
 for i in range(tSize):  
 tValue = i \* tauArg  
 uValue0 = ut0.subs(t, tValue)  
 uValue1 = ut1.subs(t, tValue)  
 res[i, 0] = (0.0, tValue, uValue0)  
 res[i, xSize - 1] = (xValueLast, tValue, uValue1)  
  
 for i in range(1, tSize):  
 print("step {0} of {1}".format(i, tSize))  
 tValue = i \* tauArg  
  
 for j in range(1, xSize - 1):  
 res[i, j] = (  
 res[i - 1, j][0],  
 tValue,  
 (1 - 2 \* lam) \* res[i - 1, j][2] +  
 lam \* res[i - 1, j - 1][2] +  
 lam \* res[i - 1, j + 1][2]  
 )  
 return res  
  
  
# Неявный метод  
def implicitMethod(u0x, ut0, ut1, aArg, hArg):  
 x = sy.Symbol('x')  
 t = sy.Symbol('t')  
 tauArg = hArg \*\* 2 / (2 \* aArg + 3)  
 tSize = math.floor(10 / tauArg) + 1  
 xSize = math.floor(1 / hArg) + 1  
 res = np.zeros((tSize, xSize), dtype='f,f,f')  
 lam = aArg \* tauArg / hArg \*\* 2  
  
 for i in range(xSize):  
 xValue = i \* hArg  
 uValue0 = u0x.subs(x, xValue)  
 res[0, i] = (xValue, 0.0, uValue0)  
  
 xValueLast = (xSize - 1) \* hArg  
  
 for i in range(tSize):  
 tValue = i \* tauArg  
 uValue0 = ut0.subs(t, tValue)  
 uValue1 = ut1.subs(t, tValue)  
 res[i, 0] = (0.0, tValue, uValue0)  
 res[i, xSize - 1] = (xValueLast, tValue, uValue1)  
  
 coeff = np.zeros(xSize - 1, dtype='f,f,f')  
  
 for i in range(1, tSize):  
 print("step {0} of {1}".format(i, tSize))  
 tValue = i \* tauArg  
  
 d = -res[i - 1, 1][2] - lam \* res[i, 0][2]  
 aj = lam / (1 + 2 \* lam)  
 bj = -d / (1 + 2 \* lam)  
  
 coeff[0] = (aj, bj)  
 for j in range(2, xSize - 2):  
 d = -res[i - 1, j][2]  
 ej = lam \* coeff[j - 2][0] - (1 + 2 \* lam)  
 aj = -lam / ej  
 bj = (d - lam \* coeff[j - 2][1]) / ej  
 coeff[j - 1] = (aj, bj)  
  
 d = -res[i - 1, xSize - 2][2] - lam \* res[i, xSize - 1][2]  
 uValue = (d - lam \* coeff[xSize - 4][1]) / (-(1 + 2 \* lam) + lam \* coeff[xSize - 4][0])  
 res[i, xSize - 2] = (hArg \* (xSize - 2), tValue, uValue)  
 for j in range(xSize - 3, 0, -1):  
 uValue = coeff[j - 1][0] \* res[i, j + 1][2] + coeff[j - 1][1]  
 res[i, j] = (hArg \* j, tValue, uValue)  
  
 return res  
  
  
x\_ = sy.Symbol('x')  
t\_ = sy.Symbol('t')  
U0x = x\_ \*\* 2  
Ut0 = 0 + 0 \* x\_  
Ut1 = 1 + 0 \* x\_  
res\_ = explicitSchema(U0x, Ut0, Ut1, 1, 0.1)  
printRes(res\_)